Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Informační technologie
Studijní obor: B2646 – Informační technologie

Optimalizace výkonu aplikační vrstvy aplikace pro práci s daty měření elektrických veličin

Optimization of Performance of Data Access Layer in Application Analysing Electrical Measurement Data

Bakalářská práce

Autor: Zdeněk Hájek
Vedoucí práce: Ing. Jan Kraus

V Liberci dne 15. května 2012

Zadání

1. Seznamte se se strukturou databáze měření ENVIS a s aktuální implementací vrstev pro přístup k datům v této databázi.

2. Vytipujte jiné vhodné technologie a postupy pro implementaci takovéto vrstvy pro aplikaci v prostředí .NET.

3. Implemetujte vybrané funkce a na vhodném vzorku dat otestujte jejich výkon s využitím vhodných nástrojů.

4. Shrňte dosažené výsledky a uveďte, zda je některá z vámi navrhovaných metod vhodnější pro výše uvedenou aplikaci.

Zdeněk Hájek

Fakulta mechatroniky, informatiky
a mezioborových studií

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/200 Sb. O právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, sem si vědom povinnosti informovat o této skutečnosti TUL; v tom případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 15. května 2012

Podpis:

Poděkování

Chtěl bych poděkovat všem, kteří mi pomáhali a podporovali po celou dobu studia a při psaní bakalářské práce. Hlavně bych chtěl poděkovat vedoucímu mé práce Ing. Janu Krausovi za časté konzultace a odpovědi na mé dotazy, které byly stěžejní při vypracovávání této bakalářské práce a bez nichž bych se neobešel. Dále bych chtěl poděkovat své rodině za psychickou a finanční podporu po celou dobu studia.

Abstrakt

Tato práce se zabývá optimalizací výkonu databázové aplikace pro práci s rozsáhlými daty. Konkrétně databázové aplikace Envis, která běžnému uživateli zobrazuje data uložená v databází tak, aby je mohl pohodlně prohlížet. Data jsou do databáze ukládány z měřicích přístrojů firmy KMB.sro. Přístroje jsou umístěny například v budovách a měří fyzikální veličiny napětí, proudů a fází na jednotlivých vodičích střídavé sítě. Z těchto naměřených fyzikálních veličin dále pak počítají hodnoty ztrátových, jalových a činných výkonů, hodnoty účiníku a další veličiny dle typu měřicího přístroje. Dále pak přístroje umí zaznamenávat různé události a výpadky. Všechny naměřené a vypočítané hodnoty si přístroj ukládá do své vnitřní paměti, odkud jsou pak data přenášena například pomocí USB nebo ethernetu do databáze.

K takto uloženým datům v databázi aktuálně přistupuje aplikace Envis pomocí vrstvy, která se dotazuje na data pomocí persistentních objektů Xpo od firmy DevExpress. Cílem této bakalářské práce je zjistit, zda neexistuje efektivnější technologie pro přístup k těmto datům v prostředí .NET. V této práci je vybrána technologie T-SQL, Entity Framework a technologie LINQ to SQL. Pomocí těchto tří technologií jsou implementovány vybrané funkce a na vhodném vzorku dat otestován jejich výkon s využitím vhodných a k tomuto účelu určených nástrojů. Dosažené výsledky výkonů vybraných funkcí jsou pak pro všechny čtyři technologie shrnuty v grafech, aby bylo zřejmé, která z uvedených technologií je nejefektivnější pro aplikaci Envis.

Abstract

This work is focused on performance optimalization of database application for work with extensive data. Specifically databas application Envis, which displays data saved in databases for regular user, so he or she can comfortably view them. Data is saved in to the database from measuring devices from company KMB.sro. Devices are placed for example in buildings and they measure physical quantity of tension, current and phases on each conductor of alternating net. From these measured physical quantities they further calculate values of power dissipation, idle power and active power, values of power factor and other quantities according to the type of measuring device. Furthermore the devices can register different events and blackouts. All measured and calculated values are saved into the devices inner memory, from where they are transfered for example by USB or ethernet into the database.

To data saved in database this way is currently accessing Envis application by layer, which is asking for data by persistent objects Xpo by DevExpress company. Goal of this bachelor work is to find out, if there is some more effective technology for access to this data in .NET environment. In this work is selected T-SQL technology, Entity Framework and LINQ to SQL technology. With help of these three technologies are implemented selected functions and on propriate sample of data tested their power with use of tools intended for this purpose. Achieved results of powers of selected functions are furthermore summarized for all four technologies in tables, so it can be obvious, which of presented technologies is most effective for Envis application.

Obsah

Seznam ilustrací

Seznam tabulek

Úvod

Důvodem vzniku této práce byl požadavek na optimalizaci výkonu vrstvy pro přístup k datům měřených veličin databázové aplikace ENVIS. K tomuto účelu je potřeba se nejprve seznámit se strukturou databáze měření ENVIS. Databáze běží na instanci Microsoft SQL Server 2008 a jedná se tedy o relační databázi.

Po seznámení se strukturou databáze je potřeba zjistit, jak je implementována vrstva pro přístup k datům. Aktuálně je tato vrstva naprogramována pomocí jazyka C# a dotazuje se na data v databázi pomocí technologie persistentních objektů Xpo firmy DevExpress. Pro optimalizaci výkonu vrstvy je potřeba vybrat některé funkce z této aktuálně implementované vrstvy a implementovat je pomocí jiných vhodných technologií v prostředí .NET Framework. První technologií, která je k tomuto účelu vybrána, je technologie T-SQL. Druhou a třetí technologií pro přístup k datům v databázi v této práci jsou objektové technologie Linq to SQL a Entity Framework. Vrstvy pro všechny tři vybrané technologie píšeme v objektovém programovacím jazyce C# ve vývojovém prostředí Visual Studio 2010. Toto vývojové prostředí nabízí výkonné nástroje pro ladění kódu, což nám značně ulehčí celý proces vývoje. Pro přístup k datům v relačních databázích pomocí .NET lze použít ještě další technologie jako například Nhibernate, ale těm se v této práci věnovat nebudeme.

Naprogramované funkce pomocí zmíněných technologií je potřeba v další části práce otestovat na vhodném vzorku dat. V případě Visual Studia 2010 testujeme implementované funkce pomocí třídy Stopwatch. SQL Server 2008 nám umožňuje využít nástroj SQL Server Profiler, díky kterému můžeme sledovat dobu vykonávání jednotlivých dotazů. Pomocí těchto diagnostických nástrojů naměříme implementované funkce pro všechny čtyry technologie přístupu k datům. Získané výsledky je potřeba porovnat, aby bylo jasné, která z technologií je pro potřeby aplikace Envis nejvýhodnější.

1 Popis nástrojů pro vývoj a měření výkonu aplikačních vrstev

1.1 Microsoft SQL Server 2008

Microsoft SQL Server ([1]) je vlajkovou lodí společnosti Microsoft. Databázový systém SQL Server obsluhuje největší databáze světa. Nabízí vysoce škálovatelnou a mimořádně přizpůsobitelnou platformu architektury dat, na které lze vybudovat libovolnou myslitelnou aplikaci.

1.2 Jazyk SQL

Neboli jazyk strukturovaných dotazů([1]) je standartním dotazovacím jazykem relačních databází. SQL Server 2008 používá jazyk T-SQL, který nabízí pestrou škálu funkcí a konstruktů pro tvorbu dotazů.

1.3 SQL Server Profiler

Verze Microsoft SQL Server 2008 nabízí grafický nástroj, který umožňuje přístup k API (Application Programming Interface) SQL Trace ([1]). Pomocí nástroje Profiler můžeme definovat události SQL Server, o kterých chceme zaznamenávat informace. Lze také určit možnosti filtrování, aby nástroj zaznamenával pouze data o vybraných událostech.

1.4 Microsoft Visual Studio 2008

Microsoft Visual Studio je vývojové prostředí (IDE) od společnosti Microsoft ([2]). Může být použito pro vývoj konzolových aplikací a aplikací s grafickým rozhraním spolu s Windows Forms aplikacemi, webovými stránkami, webovými aplikacemi a webovými službami jak ve strojovém kódu, tak ve spravovaném kódu. Podporuje jazyky prostřednictvím jazykových služeb, což umožňuje, aby editor kódu a debugger podporoval jakýkoliv programovací jazyk.

1.5 LinQ

První a nejzřejmější aplikací LinQ je dotazování do externí relační databáze([4]). LinQ pro SQL je součástí projektu LinQ, který nabízí možnost dotazování do relační databáze Microsoft SQL Serveru a také do objektového modelu vycházejícího z dostupných entit. Jinými slovy, můžete definovat množinu objektů, které představují tenkou abstraktní vrstvu nad relačními daty, a do tohoto objektového modelu se dotazovat pomocí dotazů LinQ, které se ve stroji LinQ pro SQL převádějí na odpovídající dotazy SQL.

1.6 Entity Framework

Entity Framework nabízí možnosti pro správu a dotazování do entit([4]), což je úkol, který lze provádět se standartním relačním zdrojem, avšak pouze s hlubokou abstrakcí od vrstvy fyzických dat.

1.7 Objektově relační mapování

Objektově relační mapování (ORM, O/RM nebo O/R mapování) je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Řada implementací ORM se snaží v co největší míře odstínit vývojáře od nutnosti psaní SQL dotazů a pro selekci objektů z databáze používá raději objektový přístup. Takovýto postup však zpravidla umožňuje vyhledávat objekty jen podle databázového primárního klíče, což zpravidla nestačí. Proto některé implementace ORM využívají pro selekci objektů objektový dotazovací jazyk.

Jedna z výhod odstínění od práce s SQL může být i určitá nezávislost aplikace na konkrétním databázovém systému, resp. možnost zvolit databázový systém či jiné datové úložiště tak, aby vyhovovalo konkrétním podmínkám a požadavkům.

2 Struktura databáze

Než začneme psát kód implementující datovou vrstvu aplikace Envis, musíme zjistit, jaká je struktura databáze měřicích přístrojů. Bez tohoto kroku nelze napsat správně fungující datovou vrstvu. Data z přístrojů jsou ukládány v databázi do tabulek. Jednotlivé tabulky mezi sebou udržují relace pomocí cizích a primárních klíčů. Jednotlivé typy přístrojů mají v databázi své archívy. V této práci budeme načítat data z archívu Smp odpovídající přístrojům typu SMP a SMPQ. Nejprve se tedy podíváme na data uložená v jednotlivých tabulkách a vztahy mezi nimi.

2.1 Relace mezi tabulkami

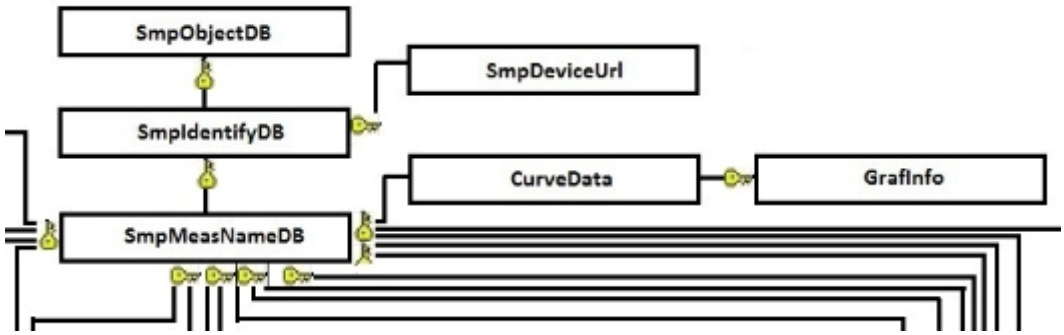
Všechny následující tabulky, začínající „SmpArchiveMain..“ obsahují data, která patří k tomuto archívu. V této práci archív označujeme Smp. Tabulky databáze, které nebudeme v práci využívat, jsou přidány do přílohy A. V příloze B je přidán obrázek, který znázorňuje celkovou struktůru databáze.

2.1.1 Tabulky obsahující informace přístrojů a jednotlivých měření

Tabulka 1: Tabulka informací k datům uložených ve výjmenovaných tabukách

Název Tabulky	Informace uložené v tabulce
SmpObjectDB	Jméno objektu pod který přístroj spadá. (Například nějaká budova)
SmpIdentifyDB	Informace o měřicím přístroji, výrobní číslo, verze software, verze hardware atd.
SmpDeviceUrl	Informace o typu připojení s přístrojem, jestli přes RS232, nebo ETHERNET, jaká rychlost atd. Pro každý přístroj je jeden a více záznamů a používá se pro opětovné připojení s přístrojem jako profil pro připojení.
SmpMeasNameDB	Jméno měření.
CurveData	Informace o jednotlivých křivkách v grafu.
CurveDataProfile	Obsahuje informace o zobrazení jednotlivých křivek z GrafUserProfile.

Pomocí tabulek *SmpObjectDB*, *SmpIdentifyDB* a *SmpMeasNameDB* (obr. 1) vytváříme v aplikaci Envis strom, pomocí něhož může uživatel vybrat měření ve vybrané budově vybraného přístroje. Tabulka *SmpObjectDB* obsahuje v sloupci informaci o názvu budovy ve které se nachází měřící přístroj a sloupec primární klíč (na obrázku žlutou barvou). Tabulka *SmpIdentifyDB* si tento primární klíč uchovává pomocí sloupce cizí klíč. Tak mezi tabulkami vzniká relace. Tabulka *SmpIdentifyDB* obsahuje informace o názvu a typu přístroje v budově a svým primárním klíčem se relací odkazuje do tabulky *SmpMeasNameDB*, která obsahuje infromace o názvu měření a odkazuje se pomocí svého primárního klíče do tabulek jednotlivých archívů. Tabulka *SmpIdentifyDB* se navíc svým primárním klíčem odkazuje do tabulky *SmpDeviceUrl*, ve které jsou informace o typu připojení databáze s měřícím přístrojem. Primární klíč tabulky *SmpMeasNameDB* kromě tabulek archívů odkazuje ještě do tabulky *CurveData*. V této tabulce společně s tabulkou *GrafInfo* jsou informace o jednotlivých křivkách grafu (barva křivky, atd..) pro potřeby aplikace Envis.



Ilustrace 1: Struktura databáze - strom

2.1.2 Tabulky s archívy naměřených dat.

Tabulka 2: Tabulka informací k datům uložených v tabulkách s archívy

Název Tabulky	Informace uložené v tabulce
SmpArchiveElmerDB	Elektroměrové záznamy ze SMP, SMPQ, a ostatních přístrojů tohoto typu.
SmpArchiveLogDB	Funguje podobně jako klasický logovací soubor a ukládá informace o událostech jako je výpadek, změna nastavení, vymazání archívu atp.
SmpArchivePqEventTrend ArchiveDB	Pro různé události zaznamenané v měřícím přístroji
SmpArchiveMainDB	Hlavní archív měření, jsou v něm uloženy naměřené hodnoty napětí, proudů a výkonů. Je použit pro SMP, SMV a podobné přístroje a vychází z něj. SIMONArchiveMainDB, který je rozšířen pro měření čtveřic proudů.
SmpArchiveMainfDB	Jsou v něm uloženy hodnoty frekvence a teploty.
SmpArchiveMainFiDB	Obsahuje informace o změřených hodnotách úhlu Fi.
SmpArchiveMainFlickerDB	Uloženy hodnoty flickeru. Blikání vnímané lidmi v závislosti na kolísání amplitudy napětí.
SmpArchiveMainIDB	Uloženy hodnoty proudů.
SmpArchiveMainPDB	Obsahuje hodnoty výkonů.
SmpArchiveMainTHDDB	Hodnoty harmonických zkreslení.
SmpArchiveMainUDB	Hodnoty napětí.
SmpArchivePmaxDB	Obsahuje informace o nejvyšším výkonu v daném měsíci.
SmpArchivePQEventDB	Všechny tabulky obsahující PQ obsahují informace o kvalitě elektrické energie. Tato tabulka obsahuje informace o různých událostech, jako výpadky, podpětí, přepětí atp.
SmpArchivePQMainDB	Desetiminutové vyhodnocení kvality elektrické energie.

Název Tabulky	Informace uložené v tabulce
SmpArchivePQOscilogramDB	Také se ukládá při událostech a ukládá průběh vln na fázích.
SmpArchiveSMProfileRecDB	Jsou zde uloženy S a M profily M profil jsou minutové záznamy napětí, proudů a výkonů, v den kdy byl naměřen maximální čtvrthodinový výkon.

Databáze obsahuje celkem 16 tabulek pro archívy různých přístrojů. Všechny tyto tabulky jsou relacemi provázány s tabulkou *SmpMeasNameDB*, která je v tabulkách archívů zastoupena svým primárním klíčem. V této práci se budeme zabývat hlavně archívy Smp pro měřicí přístroje SMP, SMV a SMPQ. Archívy těchto přístroju jsou ukládány v tabulkách *SmpArchiveMainDB*, *SmpArchiveLog*, *SmpArchivePmaxDB*, *SmpArchiveElmerDB*, *SmpArchivePQMainDB*, *SmpArchivePQEventDB*, *SmpArchivePQEventTrendArchiveDB*, *SmpArchivePqOscilogramDB*, *SmpArchiveSMProfileRecDB*. (Obr. 2)

Tabulka *SmpArchiveMainDB* obsahuje informace o tom, kdy jednotlivá měření vybraného archívu začala a kdy skončila a kolik měla vzorků. Dále pak sloupce cizích klíčů odkazujících se na primární klíče v tabulkách *SmpArchiveMainfDB, SmpArchiveMainFlickerDB, SmpArchiveMainUDB, SmpArchiveMainIDB, SmpArchiveMainPDB, SmpArchiveMainTHDDB a SmpArchiveMainFiDB*. (Obr. 2) V těchto jmenovaných tabulkách jsou jednotlivé záznamy průměrných, minimách a maximálních sdružených a fázových napětí, proudů, zaznamenaných fázích, přístrojem vypočítaných hodnot jalových, činných a zdánlivých výkonů, atd..

Ilustrace 2: Struktura databáze – tabulky Smp

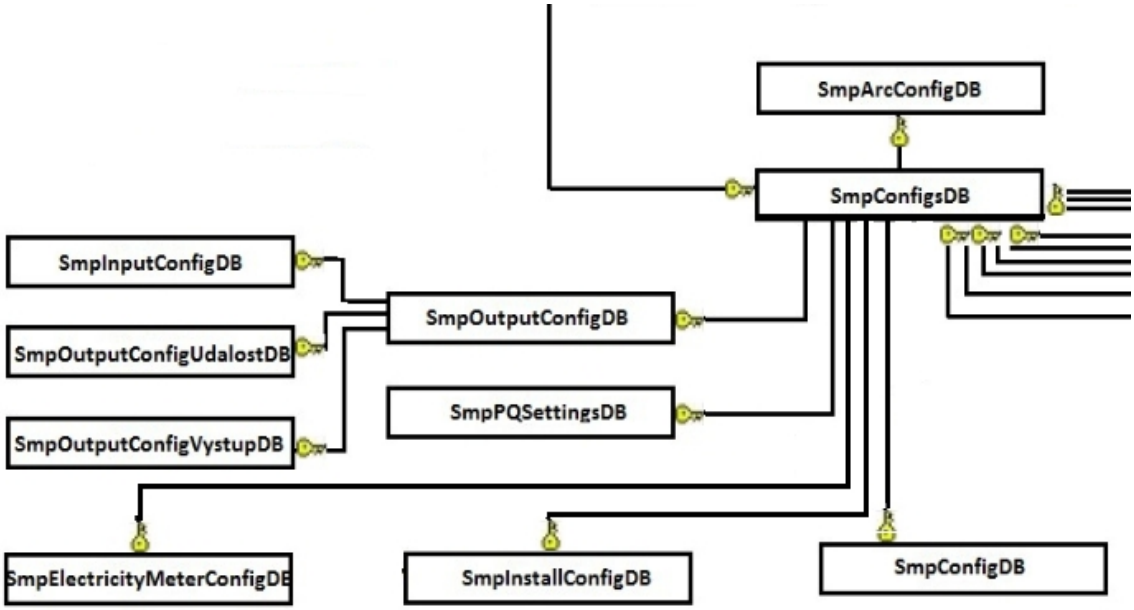
Tabulka *SmpArchiveMainDB* obsahuje cizí klíč odkazující se na primární klíč v tabulce *SmpConfigsDB*. Na primární klíč tabulky *SmpConfigsDB* se cizím klíčem odkazují všechny tabulky uchovávající archívy typu Smp a navíc se na ní odkazují i archívy pro přístroj SIMONPQ z tabulky SIMONArchiveMainDB.

2.1.3 Tabulky s informacemi o konfiguraci přístroje.

Tabulka 3: Tabulka informací k datům uložených v tabulkách s konfiguračními daty

Název Tabulky	Informace uložené v tabulce
SmpConfigDB	Obsahuje nastavení přístrojů, např. adresa přístroje, IP adresa atd.
SmpConfigsDB	Obsahuje odkazy na všechny konfigy přístrojů typu Smp…
SmpArcConfigDB	Používá se ve spojení s SIMONArchiveMainDB nebo SmpArchiveMainDB a obsahuje informace o tom, které měřené veličiny, byly z přístroje v daném archívu staženy.
SmpElectricityMeterConfigDB	Nastavení elektroměru.
SmpInputConfigDB	Nastavení vstupů.
SmpInstallConfigDB	Nastavení hodnot měřicích/převodních traf, MTN, MTP, atd.
SmpOutputConfigDB	Nastavení výstupů, souvisí s ní následující dvě tabulky.
SmpOutputConfigUdalostDB	Nastavení, na které události má výstup reagovat.
SmpOutputConfigVystupDB	Jak se má ten výstup chovat, když nastane událost, např. sepnout, rozepnout, blikat atp.
SmpPQSettingsDB	Nastavení vyhodnocení kvality elektrické energie.

Každý typ přístroje má v databázi kromě tabulek s naměřenými archívy dat navíc tabulky obsahující konfigurační data přístroje v době daného měření. Pro archívy typu Smp je to tabulka *SmpConfigsDB*. Tato tabulka v sobě uchovává cizí klíče odkazující se na primární klíče v tabulkách *SmpConfigDB, SmpArcConfigDB, SmpInstallConfigDB, SmpElectricityMeterConfigDB, SmpPQ SettingsDB* a *SmpOutputConfigDB*, která v sobě uchovává sloupce cizích klíčů od primárních klíčů tabulek *SmpInputConfigDB, SmpOutputConfigUdalostDB, SmpOutputConfigVystupDB*. (Obr. 3)



Ilustrace 3: Struktura databáze – konfigurační data přístroje

3 Aktuální implementace vrstvy pro přístup k datům v databázi

Aplikace Envis využívá pro přístup k datům v databázi rozhraní *IDataSource* z knihovny Envis.Facade firmy KMB.sro. Toto rozhraní při získávání dat z databáze používá technologii persistentních objektů Xpo od firmy DevExpress. Knihovna DevExpress.Xpo obsahuje třídy, které po jejich implementaci můžeme využít k mapování databáze do paměti.

```
/Vytvoření třídy SmpArchiveMainDB implementující XPLiteObject
 public class SmpArchiveMainDB : XPLiteObject
 {
    //Session
    public SmpArchiveMainDB(Session s) : base(s){}
    //konstruktor
    public SmpArchiveMainDB() { }
    //Proměnné persistentního objektu
    [Key(true)]
    public DateTime keyTime;
    protected int _RecordCount;
    protected DateTime _endTime;
    protected Int16 _OverflowStatus;
    protected Int16 _UnderflowStatus;
                    •
                    •
```

Ilustrace 4: Kód 1. - Vytvoření persistentního objektu SmpArchiveMainDB

Na obrázku (Obr. 4) je zobrazen začátek kódu v jazyce C#. Pomocí tohoto kódu mapujeme strukturu tabulky SmpArchiveMainDB do paměti v podobě persistentního objektu. Třída SmpArchiveMainDB implementuje třídu XPLiteObject z knihovny Xpo. Uvnitř třídy je kód vytvářející Session. Tato Session pomáhá seskupit k sobě patřící persistentní objekty. Všechny třídy implementující třídu XPLiteObject obsahují Session. Dále v kódu na obrázku máme

konstruktor vytvářející instanci třídy SmpArchiveMainDB. Pod konstruktorem už jsou jako proměnné třídy nadefinovány jednotlivé sloupce tabulky SmpArchiveMainDB. Proměnná keyTime je primárním klíčem tabulky SmpArchiveMainDB označena v kódu pomocí ([Key(true)]).

```
//Zapouzdření
    public int RecordCount
    {
      get { return _RecordCount; }
      set { SetPropertyValue<int>("RecordCount", ref _RecordCount, value); }
    }
```

Ilustrace 5: Kód 2. - Vytvoření persistentního objektu SmpArchiveMainDB

Na obrázku (Obr. 5) máme kód, který zapouzdřuje proměnnou RecordCount, takto jsou zapouzdřeny všechny proměnné daného typu pro sloupce z tabulky SmpArchiveMainDB. Pro sloupce, které obsahují cizí klíč, se definuje typ proměnné podle třídy tabulky, která obsahuje primární klíč od daného cizího klíče. Například máme k tabulce SmpArchiveMainUDB vytvořenou třídu SmpArchiveMainUDB implementující třídu XPLiteObject. Potom tedy bude v naší třídě SmpArchiveMainDB proměnná datového typu SmpArchiveMainUDB (Obr. 6).

```
protected SmpArchiveMainUDB avg_uLN
public int _avg_uLN
{
  get { return avg_uLN; } nt; }
  set { SetPropertyValue<SmpArchiveMainUDB>("_avg_uLN", ref _avg_uLN, value); }
}
```

Ilustrace 6: Kód 3. - Vytvoření persistentního objektu SmpArchiveMainDB

Tímto způsobem jsou vytvořeny třídy pro všechny tabulky z databáze a tím vytvořena struktůra databáze v paměti. Tyto třídy persitentních objektů budeme využívat i v našich implementacích *IDataSource* v technologiích LinQ, Entity Framework a T-SQL, akorát nebudeme využívat funkci Session. Pomocí Session může technologie persistentních objektů Xpo efektivně sestavovat dotazy do databáze a získaná data ukládat do připravených struktur persistentních objektů. Tyto struktury s daty vrací rozhraní *IDataSource* po načtení potřebných dat do další vrstvy, která má za úkol data zobrazit na výstupu aplikace Envis například v podobě tabulky nebo grafu. V našich implementacích rozhraní *IDataSource* budeme tedy také předávat získaná data do těchto struktur, abychom mohli data vracet v konzistentním stavu do další vrstvy a ta se tak nemusela přepisovat.

4 Jiné vhodné technologie pro implementaci vrstvy v prostředí .NET

V prostředí .NET společnosti Microsoft dnes existují různé technologie na dotazování se na data uložená v relační databázi. Tyto technologie se nazívají T-SQL, LINQ to SQL, nhibernate a Entity framework od společnosti microsoft, které můžeme použít zdarma. Pro tuto bakalářskou práci jsme vybrali technologie T-SQL, Entity Framework a LINQ to SQL.

4.1 T-SQL

Tato technologie dotazování se na data v relační databázi nabízí pestrou škálu funkcí a konstruktorů pro tvorbu dotazů. To nám umožňuje tvořit dotazy na data pro potřeby metod našeho rozhraní. Naše rozhraní dotazující se pomocí T-SQL implementuje rozhraní *IDataSource* a jmenuje se *MDataSource*.

```
SqlDataAdapter adapter = null;                  // vytvoření adaptéru
        DataSet dSet = new DataSet();           // vytvoření Datasetu
        using (SqlConnection conn = new SqlConnection(retezec))
        {
           conn.Open();                         // otevření spojení do databáze
           adapter = new SqlDataAdapter(dotaz, conn); //provede dotaz
           adapter.Fill(dSet, "Tabulky"); // výsledek v Datasetu
        }                                       // spojení do databáze ukončeno
        DataTable tab = dSet.Tables["Tabulky"]; //výsledek v tabulce
```

Ilustrace 7: Kód 4. - Načtení dat z databáze pomocí datového adaptéru

Rozhraní *MDataSource* k připojení k databázi využívá funkce knihovny System.Data.SqlClient. Pro zpracování dotazu je potřeba vytvořit datový adaptér (Obr. 7), který

spustí dotaz v databázi a načtená data uloží do tabulky v DataSetu. Klíčové slovo using vytvoří instanci SqlConnection, které předáme v proměnné typu string připojovací řetězec do databáze. Tento připojovací řetězec obsahuje název vybrané instance serveru a jméno databáze. Poté se připojení otevře a vytvoří se datový adaptér. V parametrech konstruktoru předáváme, jaký dotaz se má v jaké databází spouštět. Metoda datového adaptéru Fill poté spustí dotaz na databázi a vytvoří pro získaná data tabulku v datové sadě. Poté se ukončí blok using a spojení se serverem se ukončí. Na posledním řádku na obrázku vytváříme tabulku. Data z této tabulky v metodě přetypujeme, protože Microsoft SQL Server 2008 používá jiné typování než programovací jazyk C#. Po přetypování data ukládáme do připravených struktur persistentních objektů a ve vhodném datovém typu vracíme do vrstvy, která o ně zažádala.

4.2 LinQ

LINQ je programovací model, který zavádí dotazy jako prvořadý princip do všech jazyků Microsoft .NET([4]). Úplná podpora LINQ vyžaduje určitá rozšíření použitého programovacího jazyka, v našem případě jazyka C#. Tato rozšíření zvyšují efektivitu vývojářů a poskytují kratší, smysluplnější a jasnější syntaxi pro manipulaci s daty. LINQ představuje zajímavý krok ve vývoji současných běžných programovacích jazyků. Nabízí metodologii, která zjednodušuje a sjednocuje implementaci libovolného přístupu typu dat. Exekuční prostředí může nabídnout programu napsanému na vyšší úrovni abstrakce, na jaké se pohybuje LINQ, mnoho dalších zajímavých služeb. Dnes je významné tuto novou technologii dobře pochopit, ale zásadní vliv může získat až v budoucnu.

4.2.1 Jak LINQ pracuje

Než se začneme dotazovat do relační databáze pomocí technologie LINQ, je třeba určit, odkud se budou data získávat. K tomuto účelu obsahují knihovny Linq třídu DataContext, které

v konstruktoru při vytváření objektu této třídy předáváme v parametru připojovací řetězec k databázi. (Obr. 8). Na obrázku kromě vytváření instance objektu typu DataContext máme zobrazen ještě způsob vytvoření kolekce pro objekty typu SmpObjectDB a vytvoření instance namapované tabulky SmpObjectTable, kterou využíváme dále v dotazovacím výrazu LINQ. Takto si musíme vytvářet instance namapovaných tabulek pro všechny tabulky, ze kterých budeme získávat data.

```
//Vytvoření kolekce pro persistentní objekty typu SmpObjectDB
 List<SmpObjectDB> objekty = new List<SmpObjectDB>();
 //Vytvoření datového kontextu pro připojení k databázi
 DataContext db = new DataContext(retezec);
 //Před použitím v dotazu je třeba si mapovanou tabulku v metodě definovat
 Table<SmpObjectTable> SmpObjectTable = db.GetTable<SmpObjectTable>();
```

Ilustrace 8: Kód 5. - Vytváření kolekce pro persistentní objekty

```
//Kód v LINQ
var dotaz =
        from c in SmpIdentifyTable
        where c.objekt == p.Id
        select c;
//Kód vygenerovaný kompilátorem
var dotaz = SmpIdentifyTable
    .Where(c => c.objekt == p.Id)
    .Select(c => new { c.Id, c.objekt });
```

Ilustrace 9: Kód 6. - Kompilace dotazovacího výrazu.

Syntaxe používaná v LINQ je podobná SQL a nazývá se dotazovací výraz. Nějaký dotaz, podobný SQL a smíchaný se syntaxí programu napsaného v jiném jazyce než SQL, se obvykle

nazývá vnořené (Embedded) SQL, ale jazyky, které takové dotazy implementují, obvykle používají zjednodušenou syntaxi. Dotaz v LINQ (Obr. 9) na data v tabulce SmpIdentifyDB se v kompilátoru přeloží do tvaru, z kterého je zjevné, že kód volá členy instance objektu z předchozího volání.

```
/Mapování struktury tabulky SmpObjectDB
 [Table(Name="SmpObjectDB")]
 public class SmpObjectTable
 {
    [Column(IsPrimaryKey = true)]
    public int Id;
    [Column]
    public string objekt;
 }
```

Ilustrace 10: Kód 7. - Mapování tabulky SmpObjectDB

Where se volá na objektu SmpIdentifyTable a Select se volá na objektu navráceném klauzulí Where. Klíčové slovo var dává programu vědět, že se jedná o dotaz LINQ. To nám umožňují funkce knihovny systém.Data.Linq společnosti Microsoft. Objekt SmpIdentifyTable mapuje v LINQ strukturu tabulky SmpIdentifyDB do paměti. Aby jsme mohli takto mapovat tabulky z databáze do paměti, musíme použít knihovnu System.Data.Linq.Mapping. Pro příklad (Obr. 10) si uvedeme nadefinování struktury třídy SmpObjectTable patřícího k tabulce SmpObjectDB. V těle třídy jsou definovány datové typy jednotlivých sloupců tabulky SmpObjectDB odpovídající datovým typům v jazyce C#. Primární klíč tabulky je pak označen pomocí (IsPrimaryKey=true). Sloupce obsahující v databázi hodnoty null musíme označit otazníkem. Například public int? Id. Takto si namapujeme do paměti všechny tabulky, které budeme v našich dotazech potřebovat.

Naše třída pro rozhraní využívající technologii LINQ implementuje rozhraní *IDataSource* a v práci se jmenuje *LDataSource*. V rozhraní *LDataSource* máme namapovány všechny potřebné tabulky a využíváme v něm dříve uvedené knihovny pro práci s LINQ. Metody tohoto rozhraní mají ve svém těle nadefinován dotazovací výraz do namapovaných tabulek. Samotný SQL dotaz do databáze se vygeneruje a spustí až tehdy, když dojde na samotné získávání dat. V našem případě k tomuto dojde v cyklu foreach . Na obrázku (obr. 11) je na prvním řádku vidět vytvoření instance persistentního objektu SmpObjectDB. Tento objekt se v cyklu foreach naplní získanými daty a přidá se pomocí .Add(jm) do předem připravené kolekce objektů typu SmpObjectDB. Poté je předán vrstvě pro zpracování získaných dat.

```
SmpObjectDB jm = new SmpObjectDB(); //Xpo objekt
foreach (var c in dotaz) // zde se spustí SQL dotaz
{
   jm.Id = c.Id; //přiřazení hodnoty do objektu Xpo
   jm.objekt = c.objekt;
   objekty.Add(jm);
}
```

Ilustrace 11: Kód 8. - přiřazení výsledků dotazovacího výrazu do persistentního objektu

4.3 Entity Framework

Každý datový model je vnitřně založen na množině schémat XML a model tato schémata mapuje na trvalou fyzickou vrstvu. Uvedené soubory XML se analyzují v nástroji, který generuje odpovídající kód implementace v .NETu. Těmto procesům se říká datové modelování entit (EDM). První soubor s metadaty XML je soubor v definičním jazyce konceptuálního schématu CSDL, který v našem aplikačním prostředí definuje oblast entit. Soubor tohoto typu popisuje veškeré položky typu EntityType (konkrétní entita), EntitySet (skupina entit) a Association (vztah mezi entitami). Druhý soubor s metadaty XML je napsán v definičním jazyce schématu úložiště (SSDL) a popisuje fyzickou datovou vrstvu. Poslední soubor s metadaty XML je napsán v jazyce schématu mapování (MSL) a pouze mapuje dva předchozí definiční soubory s metadaty. Pomocí těchto XML souborů se použitím nástroje EdmGen.exe vygeneruje soubor s příponou .cs, který obsahuje kód v jazyce C# s definicemi všech typů EntityType, EntitySet a Association zadaných v souboru CSDL.

4.3.1 Dotazování do entit pomocí ADO.NET

ADO.NET Entity Framework obsahuje množinu tříd, které umožňují pracovat s entitami z vygenerovaného souboru z nástroje EdmGen.exe. Součástí je řízený poskytovatel ADO.NET pro správu entit namísto záznamů. Tento nový poskytovatel, definovaný ve jmenném prostoru System.Data.EntityClient v sestavení System.Data.Entity, obsahuje třídu EntityConnection. Ta dědí propojovací řetězec, který vyžaduje nejen databázové spojení, ale také cestu k množině souborů s metadaty v XML a typ poskytovatele, který se má použít pro fyzickou datovou vrstvu. Připojovací řetězec je uložen v souboru App.Config naší aplikace.

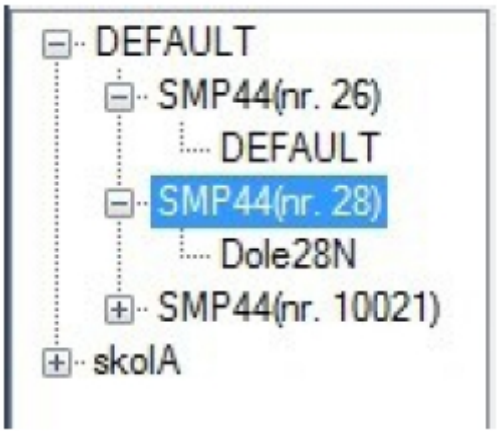
```
using (projektEntities1 db = new projektEntities1())
{
   var SmpObjectDB = db.CreateQuery<SmpObjectDB>(
      "SELECT VALUE c FROM SmpObjectDB AS c");

   foreach (var c in SmpObjectDB)
   {
      ENVIS.Model.SmpObjectDB jm = new ENVIS.Model.SmpObjectDB();
      jm.objekt = c.objekt;
      jm.Id = c.Id;
      objekty.Add(jm);
   }
}
```

Ilustrace 12: Kód 9. - Dotaz do databáze pomocí Entity Framework

V klíčovém slovu using jazyka C# se vytvoří instance projektEntities1 (obr. 12). Tato instance popisuje soubory modelu, který funguje jako kontejner pro všechny instance typu EntitySet. Dotaz do entit vytváříme voláním generické metody CreateQuery objektu typu projektEntities1. Metoda CreateQuery je definována v základní třídě ObjectContext a nevrací výsledek ihned. Namísto toho vytváří dotazovací strom, který se rozřeší v okamžiku použití.

Výsledkem dotazu je množina objektů typu DbDataRecords, kde každý obsahuje jeden sloupcový řádek reprezentující jednu instanci typu SmpObjectDB. Tato instance je v cyklu foreach vkládána do kolekce persistentních objektů SmpObjectDB. Naše rozhraní dotazující se pomocí EntityFramework implementuje rozhraní *IDataSource* a jmenuje se *EFDataSource*.

5 Implementace vybraných funkcí



Ilustrace 13: Strom v komponentě treeView

V naší práci jsme pro optimalizaci výkonu databázové aplikace vybrali funkce rozhraní IDataSource jménem GetObjects, GetIdents, GetRecords, ExistSomeArchives a GetRows. Metody GetObjecs, GetIdents a GetRecords využívá vrstva, která metody volá, při vykreslování stromu (Obr. 13) sloužícího uživateli k výběru měření přístroje umístěného v budově. Na obrázku vidíme jako kořenové uzly budovy. Potomkem kořenového uzlu je uzel přístrojů. Jedna budova totiž může obsahovat více přístrojů. A potomkem uzlu přístrojů je uzel měření. Jeden přístroj může obsahovat více měření.

Metoda ExistSomeArchives, volaná na instanci objektu rozhraní (MDataSource, LDataSource, IDataSource, EFDataSource), má za úkol zjistit, které archívy obsahují data a které jsou prázdné. Archívy s daty se pak zapíší do komponenty listBox, odkud je umožněno uživateli aplikace vybrat archív měření s naměřenými hodnotami.

Metoda GetRows má za úkol načítat data z vybraných archívů. Získaná data pak vrací v kolekci datového typu XPCollection do vrstvy, kde se nachází instance objektu, která metoduGetRows zavolala. V následující části práce si ukážeme, co tyto jednotlivé metody dělají a jakým způsobem je budeme implementovat.

5.1 GetObjects

Tato metoda slouží k získání dat z tabulky SmpObjectDB. V těle metody se vytvoří kolekce persistentních objektů SmpObjectDB pomocí třídy List knihovny System.Colection.Generic. Do této kolekce budeme vkládat jednotlivé řádky získané z tabulky SmpObjectDB v podobě persistentního objektu typu SmpObjectDB. Poté pomocí funkce ToArray() vrátíme data uložená v kolekci jako pole persistentních objektů SmpObjectDB. Řádky tabulky SmpObjectDB představují budovy, kde jsou umístěny měřící přístroje.

5.2 GetIdents

Tato metoda je určena k získání dat z tabulky SmpIdentifyDB. Jako parametr přebírá metoda GetIdents persistentní objekt typu SmpObjectDB jehož Id budeme potřebovat v našich dotazech na data z tabulky SmpIndetifyDB. Tato tabulka je v relaci s tabulkou SmpObjectDB. Stejně jako v metodě GetObjects vkládáme získaná data do kolekce. Kolekce seskupuje persistentní objekty typu SmpIdentifyDB. Po jejím naplnění všemi získanými daty použitím funkce ToArray() na kolekci vrací metoda GetIdents pole persistentních objektů SmpIdentifyDB.

5.3 GetRecords

Metoda GetRecors má za úkol získat data z tabulky SmpMeanNameDB. Parametrem je jí předávána hodnota persistentního objektu SmpIdentifyDB. Tento persistentní objekt má ve vlastnosti Id hodnotu primárního klíče tabulky SmpIdentifyDB z databáze. Hodnota Id tvoří kritérium v dotazech na data z tabulky SmpMeasNameDB. Metoda GetRecords vrací pole persistentních objektů SmpMeasNameDB.

5.4 ExistSomeArchives

Tato metoda vrací hodnotu true, pokud existují data v archívu, který je předán metodě ExistSomeArchives jako parametr. Archív je metodě předán jako objekt ArchDescriptionDB z knihovny ENVIS.Model. Metoda v parametru přebírá navíc hodnotu persistentního objektu SmpMeasNameDB, který obsahuje jméno uživatelem vybraného měření z komponenty treeView zobrazující strom. V této práci metoda ExistSomeArchives zjišťuje existenci dat v archívu Main Archive, Elmer Archive, PQ Oscillogram a PQ Event Trend Archive. Tyto archívy se týkají hlavně tabulek SmpArchiveMainDB, SmpArchiveElmerDB, SmpArchivePqOscilogramDB a SmpArchivePqEventTrendArchiveDB. V metodě ExistSomeArchives máme tedy 4 bloky kódu oddělené pomocí přepínače switch jazyka C#. Tento switch od sebe odděluje bloky pro jednotlivé archívy podle vlastnosti Name instance objektu ArchDescriptionDB.

5.5 GetRows

Metoda GetRows slouží k načtení dat z databáze uživatelem vybraných archívů z komponenty listBox. Metoda vrací kolekci XPCollection z knihovny DevExpress.XPO. Jako parametr je metodě GetRows předán persistentní objekt SmpMeasNameDB, který udržuje informaci o uživatelem vybraném měření v komponentě treeView. Hodnota vlastnosti Id persistentního objektu SmpMeasNameDB tvoří kritérium v dotazech spouštěných na databázi. Dále je metodě předán jako parametr objekt ArchDescriptionDB udržující informaci o tom, který archív byl vybrán uživatelem v komponentě listbox. Metoda GetRows stejně jako metoda ExistSomeArchives má ve svém těle přepínač switch. Přepínač switch rozhodne na základě vlastnosti Name objektu ArchDescriptionDB, na kterou tabulku se budeme v databázi dotazovat.

5.5.1 Metoda SmpConf

Metoda SmpConf získává z databáze data o konfiguraci přístroje v době měření záznamu z archívu. Parametrem přebírá hodnotu typu int představující hodnotu ve sloupci conf vybraného archívu. Na výstupu vrací persistentní objekt SmpConfigDB. V těle metody se nejdříve spouští dotaz do tabulky SmpConfigsDB. Jako kritérium se volí hodnota typu int předána metodě SmpConf jako parametr. Výsledkem dotazu bude řádek, který obsahuje hodnoty typu int, což jsou kriteria dotazů do tabulek SmpArcConfigDB, SmpConfigDB, SmpElectricityMeterConfigDB, SmpInstallConfigDB, SmpOutputConfigDB a SmpPQSettingsDB. Výsledky dotazů do těchto tabulek procházíme cyklem for a zapíšeme je do persistentního objektu SmpConfigsDB, který metoda SmpConf vrací zpět. Dotaz na data v tabulce SmpOutputConfigDB získává data i z tabulek SmpInputConfigDB, SmpOutputConfigVystupDB a SmpOutputConfigUdalostDB.

5.5.2 Main Archive

Když se hodnota vlastnosti Name objektu ArchDescriptionDB rovná řetězci "Main Archive", tak se spustí dotaz na tabulku SmpArchiveMainDB v databázi. Tato tabulka ve svých sloupcích obsahuje cizí klíče udržující relaci na tabulky naměřených veličin. Kriterium dotazu do archívu (Main Archive) říká, že hodnota primárního klíče keymeasName tabulky SmpArchiveMainDB musí být rovna hodnotě vlastnosti Id persistentního objektu SmpMeasNameDB.

Získaná data jsou procházena for cyklem. V jedné iteraci cyklu for jsou sloupce řádků přiřazeny do odpovídajících vlastností persistentního objektu SmpArchiveMainDB, který je poté vložen do kolekce XPCollection. Kolekce XPCollection je po proběhnutí všech iterací cyklu for navrácena do vrstvy, která metodu GetRecords na instanci rozhraní zavolala.

Do vlastnosti conf persistentního objektu SmpArchiveMainDB přiřazujeme v každé iteraci cyklu for hodnotu persistentního objektu SmpConfigsDB. Persistentní objekt SmpConfigsDB získáváme z metody SmpConf, kterou jsme si k tomuto účelu vytvořili. Metodě SmpConf předáváme v iteraci cyklu for hodnotu sloupce conf z výsledku dotazu. Proto v cyklu for ukládáme hodnotu conf do proměnné. V další iteraci cyklu se porovná proměnná z předchozí iterace s novou hodnotou conf. Pokud se hodnoty nerovnají, tak se znovu volá metoda SmpConf s novým parametrem. Když se hodnoty rovnají, tak využijeme hodnotu instance persistentního objektu SmpConfigsDB získanou z předchozího volání metody SmpConf. Tímto postupem můžeme zkrátit čas vykonávání metody GetRows, když více řádků za sebou z výsledku dotazu ve sloupci conf obsahuje stejnou hodnotu.

5.5.3 Elmer Archive

Vlastnost Name objektu ArchDescriptionDB rovna hodnotě řetězce "Elmer Archive" spouští dotaz na data v tabulce SmpArchiveElmerDB. Dotaz má jako v případě Main Archive zadané kritérium, že hodnota keymeasName se musí shodovat s hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB získaného z parametru metody GetRecords. Data získaná z dotazu jsou v cyklu for přiřazena do instance persistentního objektu SmpArchiveElmerDB. Sloupec conf spouští metodu SmpConf stejným způsobem, jako v případě Main Archive. Výsledná instance persistentního objektu SmpArchiveElmerDB je vložena do kolekce XPCollection. Ta je po doběhnutí všech iterací cyklu for vrácena z metody GetRecords vrtsvě, která metodu na instanci rozhraní zavolala.

5.5.4 Pq Oscillogram

Dotaz do tabulky SmpArchivePqOscilogramDB se spustí, když se hodnota vlastnosti Name persistentního objektu ArchDescriptionDB rovná řetězci "Pq Oscillogram". Kritérium dotazu je tvořeno hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB rovnající se sloupci keymeasName v tabulce SmpArchivePqOscilogramDB. Výsledky dotazu jsou v cyklu for po řádku zapsány do instance persistentního objektu SmpArchivePqOscilogramDB,

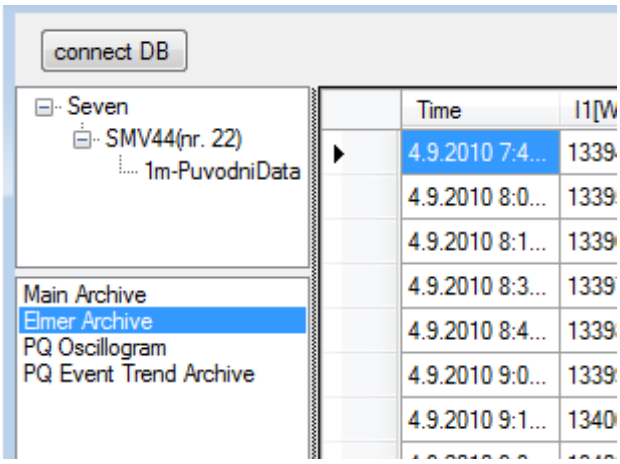
který je poté vložen do kolekce XPCollection. Na základě hodnot ve sloupci conf se v jednotlivých iteracích cyklu for spouští metoda SmpConf jako v předchozích případech. Po doběhnutí všech iterací cyklu for metoda vrací kolekci XPCollection zpět do vrstvy, která zavolala metodu GetRecords.

5.5.5 PQ Event Trend Archive

Když se hodnota vlastnosti Name objektu ArchDescriptionDB rovná řetězci "PQ Event Trend Archive", tak se spouští dotaz do tabulky SmpArchivePqEventTrendArchiveDB. Tento dotaz má stejně jako předchozí tři dotazy kritérium říkající, že ve výsledku budou pouze řádky, které mají ve sloupci keymeasName hodnotu shodnout s hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB. Výsledky dotazu přiřazujeme v cyklu for do instance persistentního objektu SmpArchivePqEventTrendArchiveDB, který poté vkládáme do kolekce XPCollection, která je metodou GetRecords vrácena. Metoda SmpConf je spouštěna dle potřeby na základě hodnot ve sloupci conf tabulky SmpArchivePqEventTrendArchiveDB.

6 Testování výkonu metod

Metody testujeme v aplikaci (Obr. 14), kterou jsme si k tomuto účelu vytvořili. Na obrázku nahoře vlevo je komponenta button, která aktivuje dialogové okno sloužící k zadání cesty k databázi. Pod komponentou button je komponenta treeView, ve které je generován strom sloužící k vybrání přístroje a názvu měření. Pod touto komponentou se nachází komponenta typu listBox, ve které se po vybrání měření v komponentě treeView objeví dostupné archívy, které pro vybrané měření byly přístrojem měřeny. Vpravo na obrázku je komponenta dataGridTable, ve které se zobrazují data vybraného archívu z komponenty listBox.



Ilustrace 14: Testovací aplikace

6.1 StopWatch

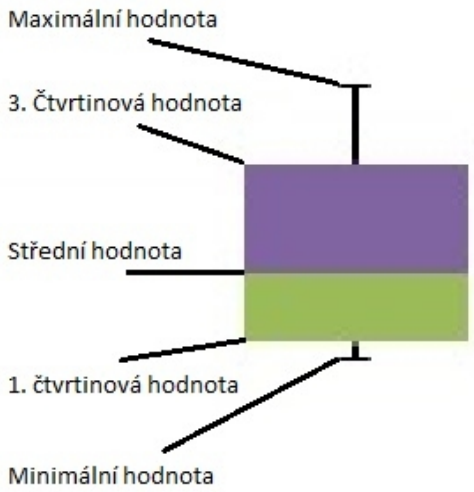
Třída StopWatch se nachází v jmeném prostoru System.Diagnostics a je vhodným prostředkem pro měření času stráveného v metodách. Pro každou naší metodu naměříme 100 vzorků času stráveného v metodách. Metody měříme (obr. 15) vložením měřené metody mezi metody Start() a Stop() vytvořené instance StopWatch. Výsledek se pomocí vlastnosti

Elapsed zapíše do proměnné typu TimeSpan, ze které lze získat čas ve formátu, který nám nejvíce vyhovuje.

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Start();
Thread.Sleep(10000);
stopWatch.Stop();
// Get the elapsed time as a TimeSpan value.
TimeSpan ts = stopWatch.Elapsed;
```

Ilustrace 15: Kód 10. - StopWatch – způsob měření

Ze získaných záznamů určíme minimální, maximální a střední hodnotu. Tyto údaje přehledně zobrazíme v tabulkách a grafech. Nejvýhodnější bude používat graf typu BoxPlot(obr. 16).



Ilustrace 16: BoxPlot

6.2 Sql Server Profiler

Pomocí nástroje SQL Server Profiler sledujeme, jaké procesy a dotazy se spouští na databázovém stroji během vykonávání měřené metody. Získaná data trasování jsou ukládána

do tabulek, které můžeme využívat pro další zpracování. Tyto tabulky trasování jsou uloženy v databázi master na naší instanci SQL server 2008. Na obrázku (Obr. 17) vidíme vytvoření pohledu "readsSUM", který vytvoří tabulku o 5 řádcích. Na každém řádku bude hodnota součtu všech hodnot ze sloupce Reads určené trasovací tabulky (MdataSourceGetObjetct1..5). To nám umožňuje agregační funkce SUM v SQL dotazech. Pomocí UNION ALL se pak výsledky jednotlivých dotazů v pohledu spojí do tabulky o 5 řádcích.

```
CREATE VIEW ReadsSUM
AS
SELECT SUM(Reads) as reads FROM [master].[dbo].[MDataSourceGetObjects1]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects2]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects3]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects4]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects5]
```

Ilustrace 17: Kód 11. - Pohled SQL vytvořený pro získání dat z trasovacích tabulek

Do pohledu ReadSUM se poté budeme dotazovat SQL dotazem (Obr. 18), kterým zjistíme směrodatnou odchylku, průměrný součet a rozptyl. Stejně jako pro sloupec Reads budeme vytvářet pohled a SQL dotaz i na sloupec Duration.

```
SELECT AVG (Reads) as prumerny_soucet, stdev(Reads) as smerodatna_odchylka,
                (stdev(Reads)*stdev(Reads)) as rozptyl FROM ReadsSUM
```

Ilustrace 18: Kód 12. - Dotaz do pohledu

```
select count(*) FROM dbo.MDataSourceGetObjects1 where EventClass = '14'
```

Ilustrace 19: Kód 13. - SQL dotaz do trasovací tabulky pro zjištění počtu spojení s databází

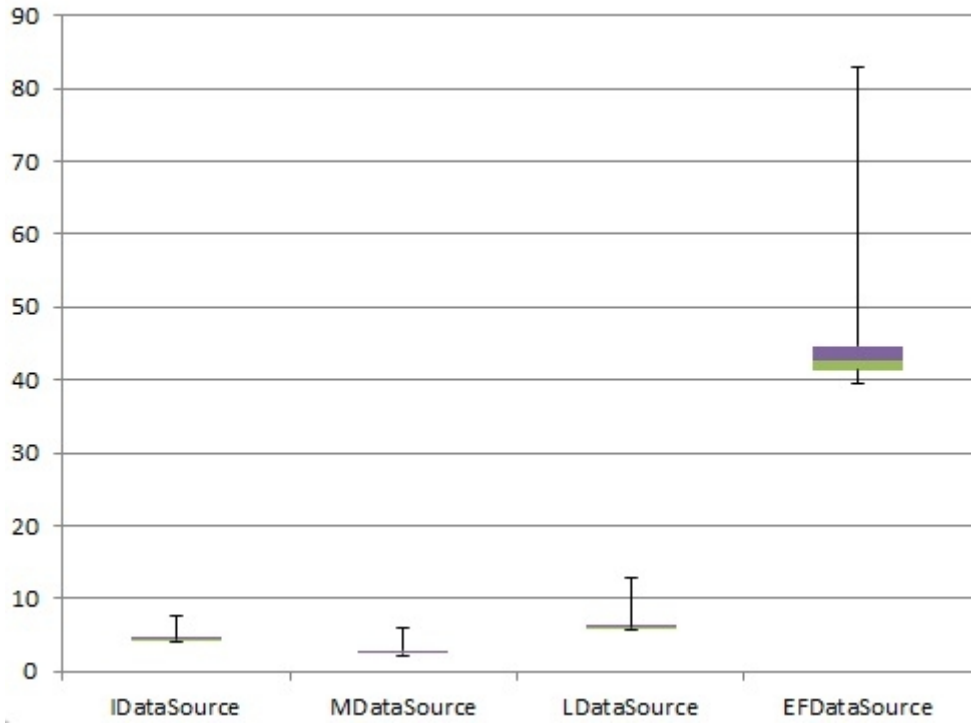
Sloupec Reads mapovacích tabulek z SQL Profileru obsahuje informaci o tom, kolikrát bylo během zpracovávání nějaké operace z databáze čteno. Sloupec Duration udržuje informaci o době strávené vykonáváním dotazu na databázovém stroji. Pro zjištění kolikrát spuštěná měřená metoda otevřela spojení do databáze použijeme SQL dotaz (Obr. 19) do získané trasovací tabulky. Tento SQL dotaz má kritérium, které říká, že sloupec EventClass musí být roven hodnotě 14. SQL dotaz využívá agregační funkce COUNT a vrací součet všech řádků vyhovujících zmíněnému kritériu. Získaná hodnota z tohoto SQL dotazu představuje počet otevřených spojení do databáze v průběhu zpracovávání měřené metody.

Nyní se podíváme na získané hodnoty v implementovaných metodách. Tyto získané hodnoty nám umožní rozpoznat, které z uvedených rozhraní pro načítání dat z relační databáze je nejvýhodnější.

7 Shrnutí výsledků

7.1 Metoda GetObjects

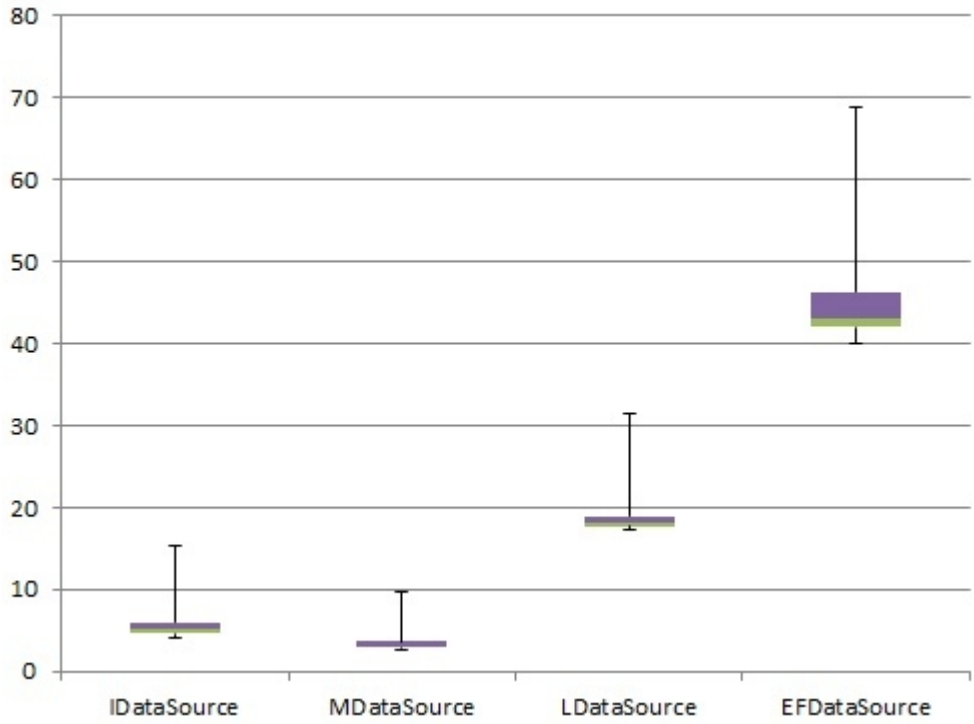
Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*. V tomto rozhraní trvá běh metody při prvním načtení průměrně 17 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 4905 ms. Po prvním načtení, když už jsou data načtena ve vnitřní paměti, se nejrychlejí načítají data metodou *MDataSource*, která je vrací průměrně za 2,73 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 45,32 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 20). Na ose y je uveden název měřeného rozhraní a na ose x je uveden čas v milisekundách. V příloze C jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 20: Graf 1: Metoda GetObject

7.2 Metoda GetIdents

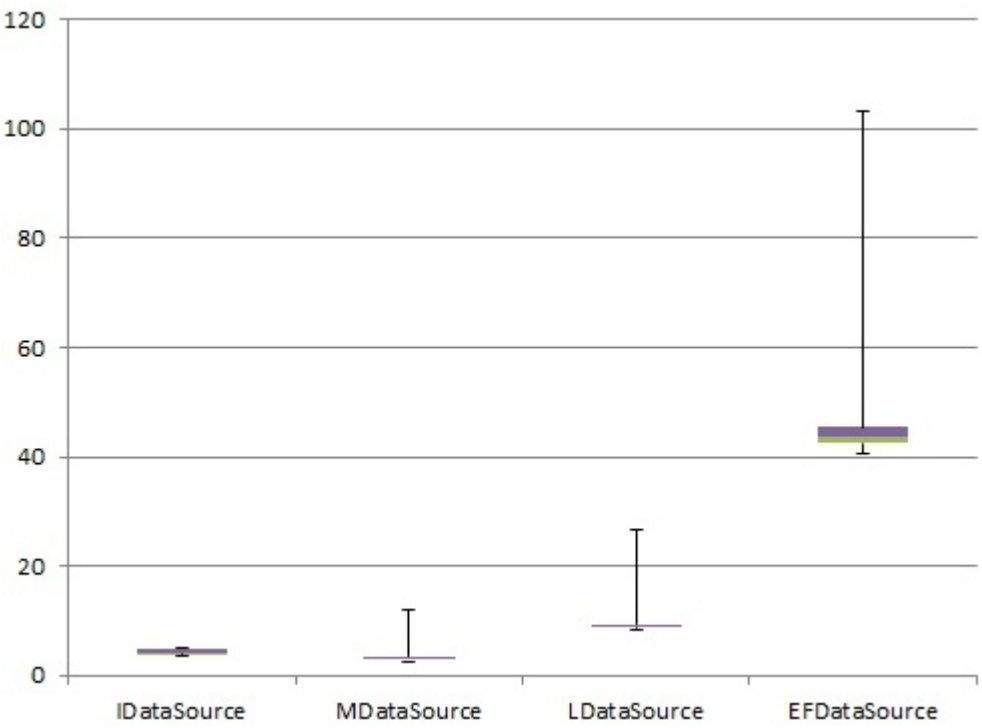
Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*, kde načtení trvá průměrně 21 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 1698 ms. Po prvním načtení se nejrychlejí načítají data metodou *MDataSource*, která je vrací průměrně za 3,54 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 45,31 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 21). V příloze D jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 21: Graf 2.: Metoda GetIdents

7.3 Metoda GetRecords

Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*, kde načtení trvá průměrně 10 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 4657 ms. Po prvním načtení se nejrychlejí načítají data metodou *MDataSource*, která je vrací průměrně za 3,46 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 45,18 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 22). V příloze E jsou tabulky s naměřenými daty výkonu této metody.

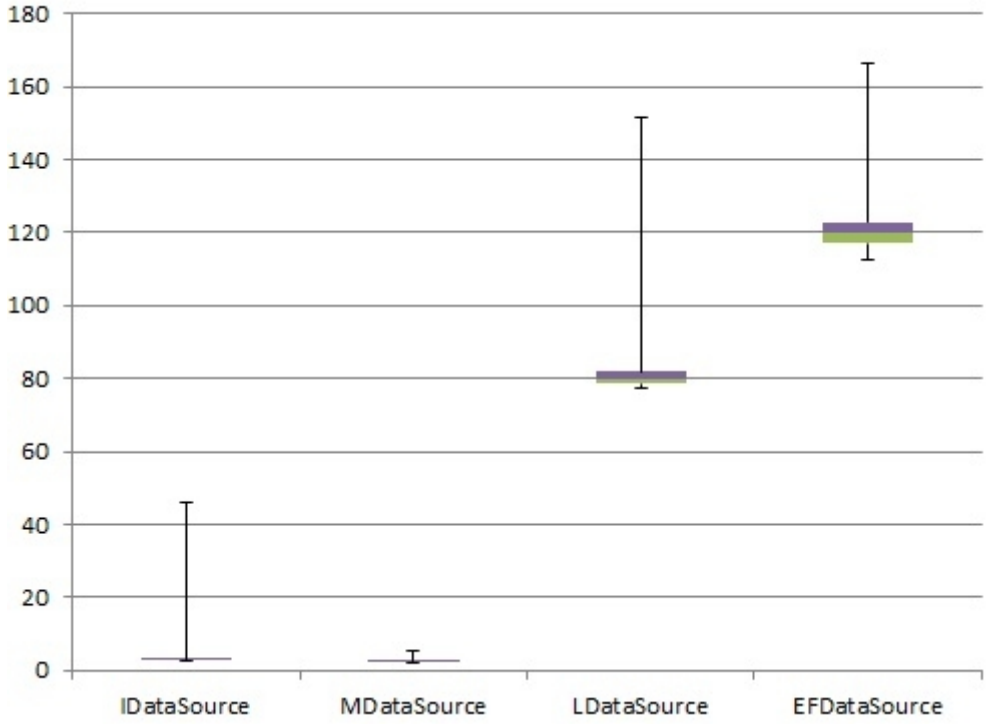


Ilustrace 22: Graf 3: Metoda GetRecords

7.4 Metoda ExistSomeArchives

7.4.1 Archive Main

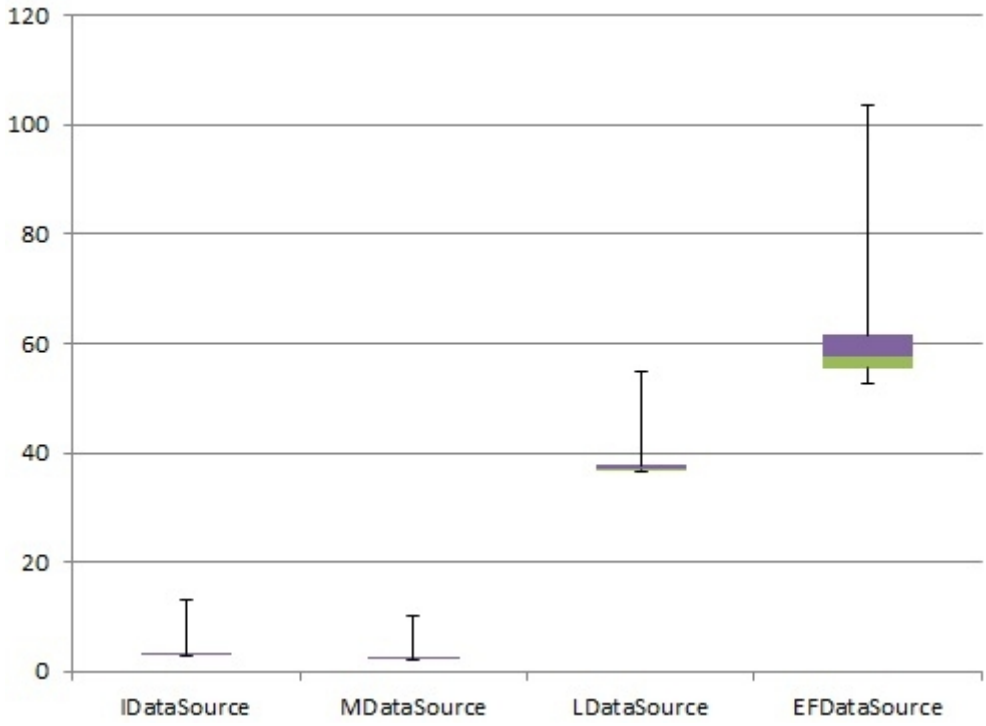
První načtení dat je nejrychlejší v metodě *IDataSource*, kde trvá průměrně 50 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 1444 ms. Po prvním načtení je nejrychlejší metoda v rozhraní *MDataSource*, která data vrací průměrně za 2,71 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 120,82 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 23). V příloze F jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 23: Graf 4: Metoda ExistSomeArchives - Main Archiv

7.4.2 Elmer Archive

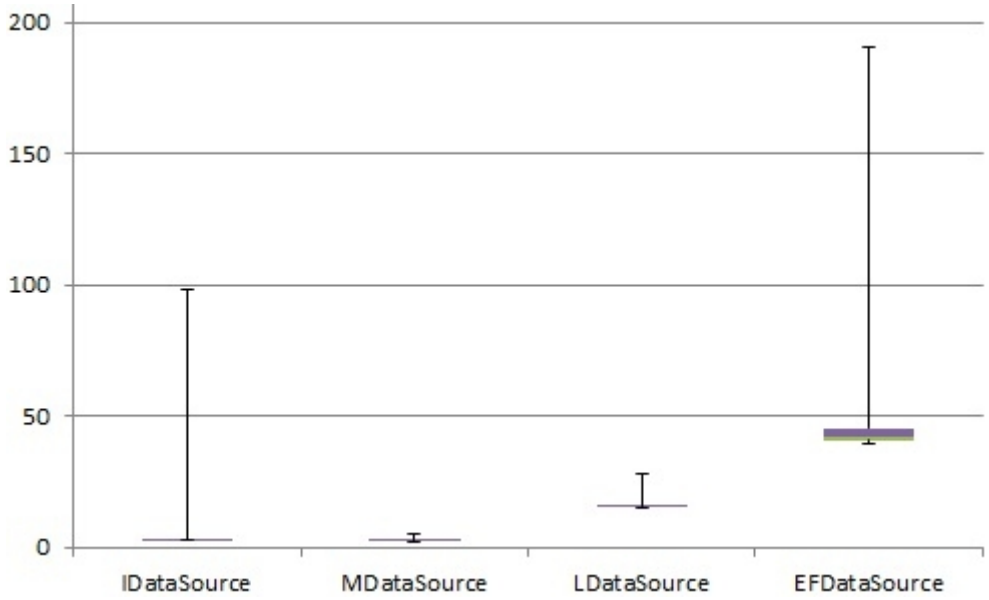
První načtení dat je nejrychlejší v metodě *IDataSource*, kde trvá průměrně 19,58 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 279 ms. Po prvním načtení je nejrychlejší metoda v rozhraní *MDataSource*, která data vrací průměrně za 2,69 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 60,12 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 24). V příloze G jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 24: Graf 5: Metoda ExistSomeArchives - Elmer Archiv

7.4.3 Pq Oscillogram

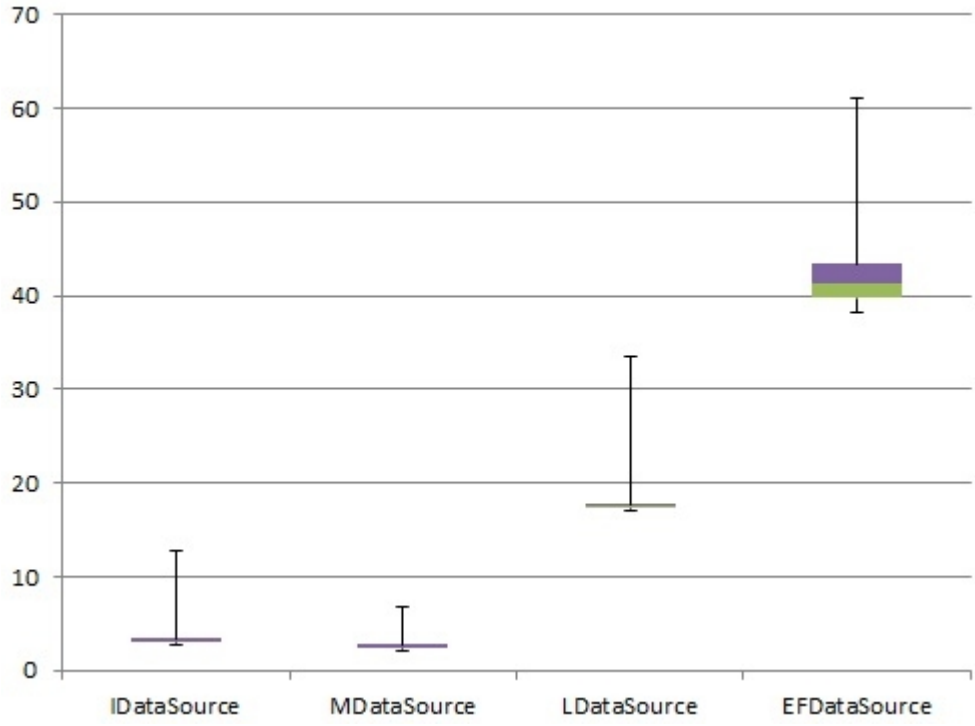
Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*, kde načtení trvá průměrně 21,8 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 339 ms. Po prvním načtení se nejrychleji načítají data metodou *MDataSource*, která je vrací průměrně za 2,48 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 47,7 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 25). V příloze H jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 25: Graf 6: Metoda ExistSomeArchives - Pq Oscillogram

7.4.4 Pq Event Trend Archive

Při prvním načtení dat je opět nejrychlejší metoda v rozhraní *IDataSource*, která načítá data průměrně 25,5 ms. Nejpomalejší je pak znovu v rozhraní *EFDataSource*, kde trvá průměrně 174 ms. Po prvním načtení se nejrychleji načítají data metodou *MDataSource*, která je vrací průměrně za 2,74 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 42,6 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 26). V příloze I jsou tabulky s naměřenými daty výkonu této metody.



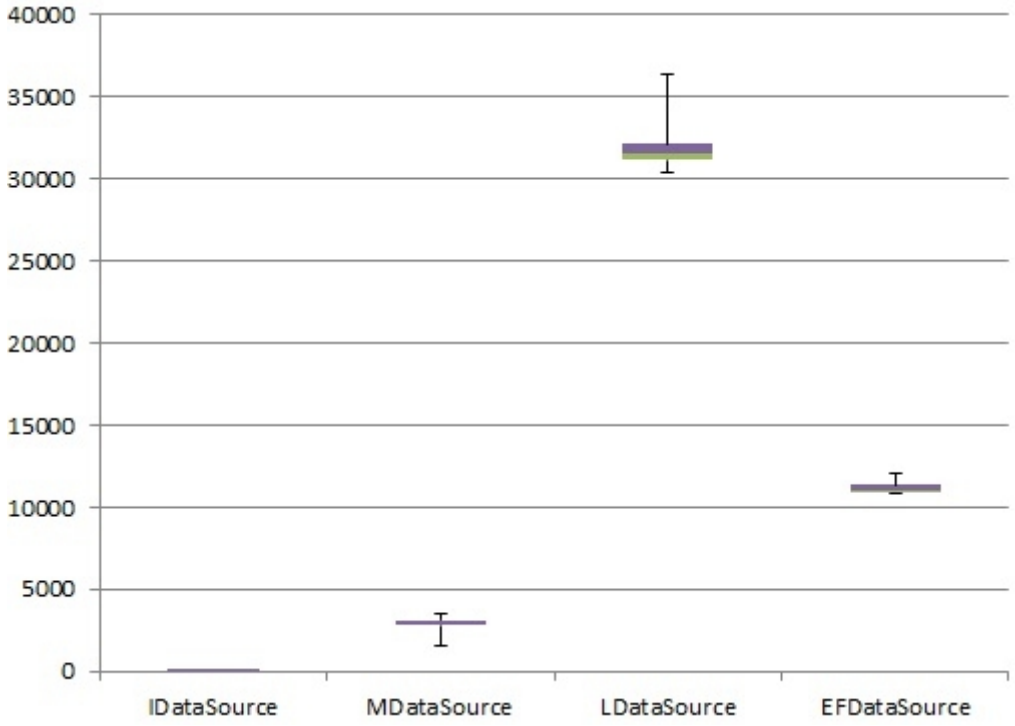
Ilustrace 26: Graf 7: Metoda ExistSomeArchives - Pq Event Trend Archive

7.5 GetRows

7.5.1 Main Archive

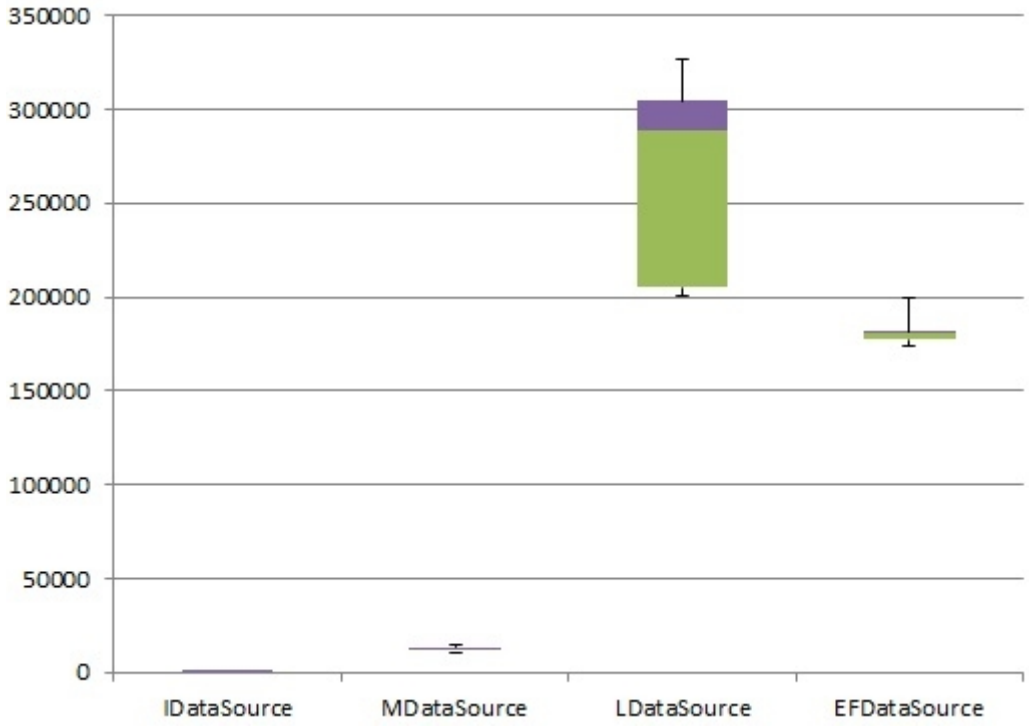
Tento archív má 251 sloupců po spojení se všemi tabulkami obsahujícími hodnoty naměřených dat. Jedná se tak o největší množství dat, které v této práci chceme načítat. V této vrstvě se spustí dotaz na data z tabulky SmpArchiveMainDB v cyklu foreach, který řádek po řádku čte z objektu XPBaseCollection. Výsledek tohoto dotazu se uloží v paměti do persistentního objektu SmpArchiveMainDB. Místo cizích klíčů jsou přiřazeny odkazy na další persistentní objekty (Např. SmpArchiveMainUDB, SmpArchiveMainIDB, atd..). Pomocí Session technologie Xpo se udržuje v paměti struktura persistentního objektu SmpArchiveMainDB s ostatními persistentními objekty. V iteraci cyklu foreach, vypisující řádek tabulky SmpArchiveMainDB do komponenty dataGridTable, se teprve spouští dotazy do tabulek obsahujících primární klíče od cizích klíčů v tabulce SmpArchiveMainDB. To je velice výhodné, protože se spouští dotazy pouze do tabulek, jejichž data chceme v komponentě dataGridTable zobrazit. Metody GetRows rozhraní *MDataSource, LDataSource a EFDataSource* jsou tedy nevýkonné, protože musejí načítat všech 251 sloupců archívu Main.

Při prvním načtení 100 řádků dat z Main archive je nejrychlejší metoda v rozhraní *IDataSource*, kde načtení trvá průměrně 1820,02 ms. Nejpomalejší je metoda v rozhraní *LDataSource*, které vrácení dat trvá průměrně 48596,25 ms. Po prvním načtení se nejrychleji načítají data metodou *IDataSource*, která je vrací průměrně za 38,11 ms. Nejpomaleji vrací data metoda *LDataSource* průměrně za 31961,2 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 27). V příloze J jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 27: Graf 8: Metoda GetRows - MainArchive - 100 řádků

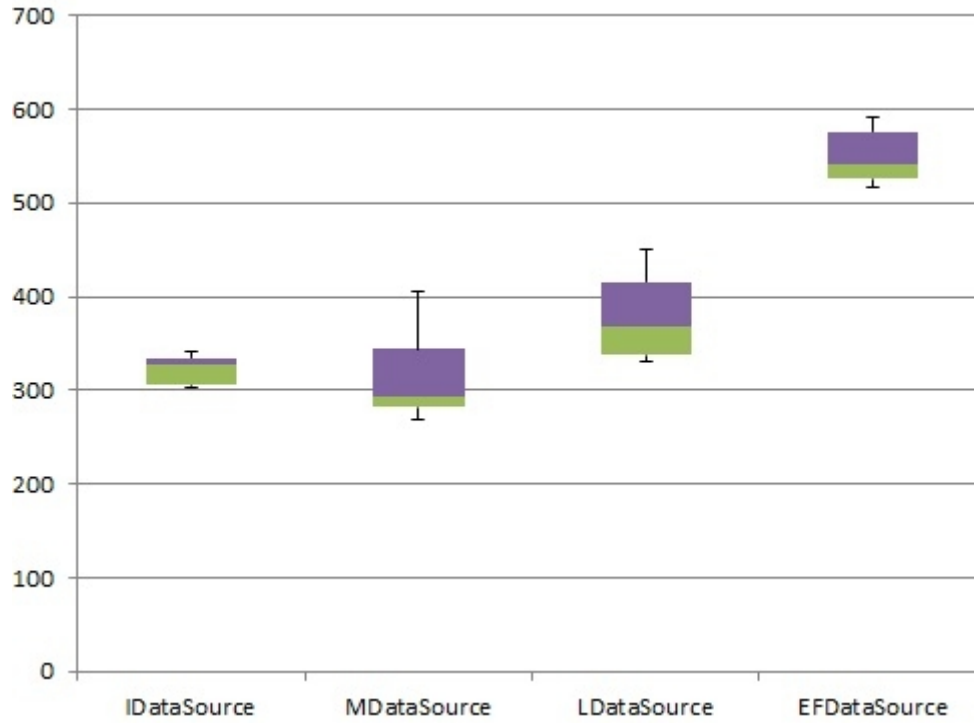
Při prvním načtení 1000 řádků dat z Main archive je nejrychlejší metoda v rozhraní *IDataSource*, kde načtení trvá průměrně 16336,67 ms. Nejpomalejší je metoda v rozhraní *LDataSource*, které vrácení dat trvá průměrně 275742,9 ms. Po prvním načtení se nejrychleji načítají data metodou *IDataSource*, která je vrací průměrně za 350,75 ms. Nejpomaleji vrací data metoda *LDataSource* průměrně za 261503,8 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 28). V příloze K jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 28: Graf 9: Metoda GetRows - Main Archive - 1000 řádků

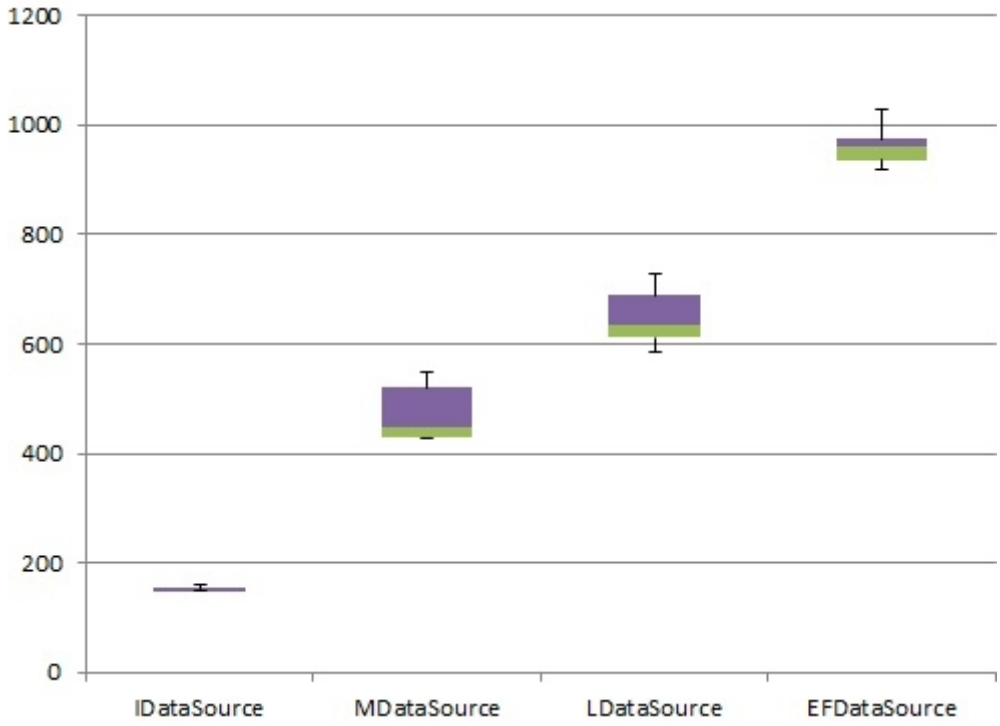
7.5.2 Archive Elmer

Při prvním načtení 1000 řádků záznamu pracuje nejrychleji metoda rozhraní *MDataSourc*e a to za průměrný čas 333 milisekund. Nejpomalejší je pak čas metody v rozhraní *EFDataSource*, který trvá pro 1000 záznamů 2306 milisekund. Po prvním načtení se nejrychleji načítají data metodou *MDataSource*, která je vrací průměrně za 315,64 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 551,76 ms. Výsledky měření metody ve všech implementovaných vrstvách pomocí třídy StopWatch jsou shrnuty v grafu (obr. 29). V příloze L jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 29: Graf 10: GetRows - Elmer Archive - 1000 řádků

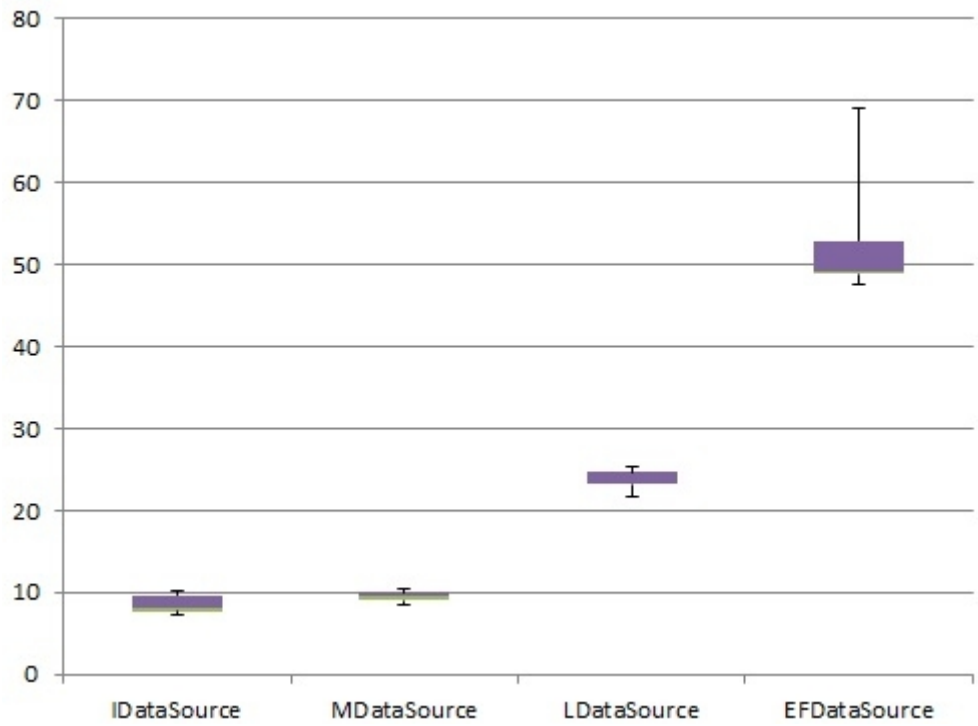
V případě 2000 a více načítaných řádcích bude nejrychlejší vždy metoda v rozhraní *IDataSource*. Nejpomaleji jsou vráceny data z rozhraní *EFDataSource*. Při prvním načtení 5000 řádků vrátí metoda *IDataSource* tyto data průměrně za 148,9 ms. Rozhraní *EFDataSource* stejné množství dat vrací nejpomaleji ze všech implementovaných rozhraní, a to v čase 964,54 ms. Výsledky měření metody pro 5000 záznamů z archivu Elmer ve všech implementovaných vrstvách pomocí třídy StopWatch jsou shrnuty v grafu (obr. 30). V příloze M jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 30: Graf 11: Metoda GetRows - Elmer Archive - 5000 řádků

7.5.3 Archiv Pq Oscillogram

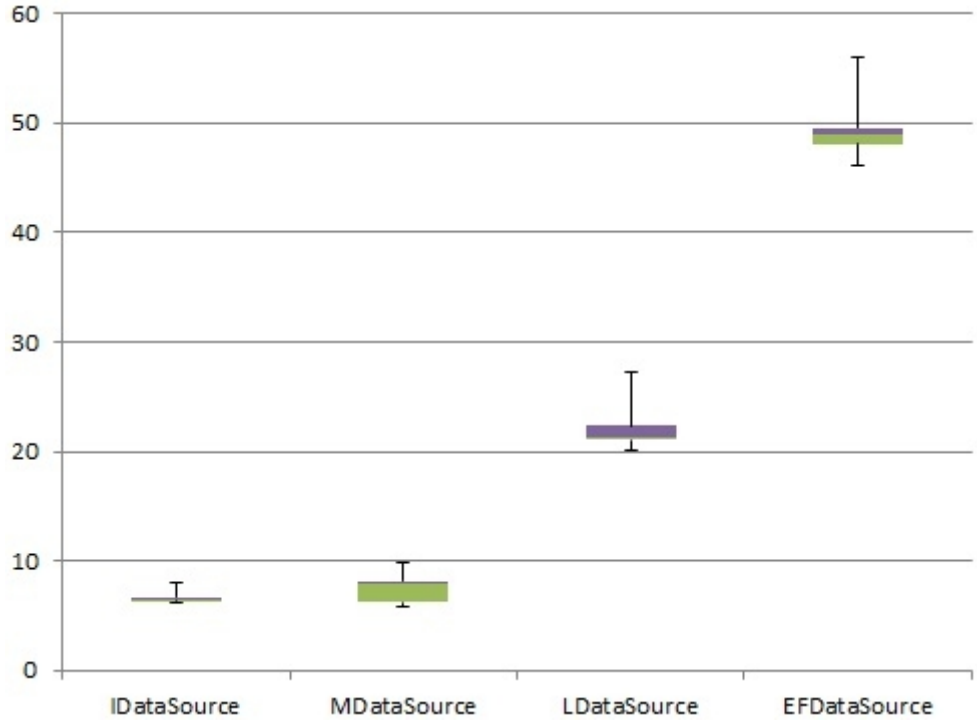
Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*, která načítá data průměrně 16,44 ms. Nejpomalejší je pak v rozhraní *LFDataSource*, kde trvá průměrně 333,9 ms. Po prvním načtení se nejrychleji načítají data metodou z *IDataSource*, která je vrací průměrně za 8,53 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 52,54 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 31). V příloze N jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 31: Graf 12: Metoda GetRows - Pq Oscillogram

7.5.4 Pq Event Trend Archive

Při prvním načtení dat je nejrychlejší metoda v rozhraní *IDataSource*, která načítá data průměrně 14,2 ms. Nejpomalejší je pak v rozhraní *EFDataSource*, kde trvá průměrně 82,86 ms. Po prvním načtení se nejrychleji načítají data metodou z *IDataSource*, která je vrací průměrně za 6,63 ms. Nejpomaleji vrací data metoda *EFDataSource* průměrně za 49,21 ms. Výsledky měření všech vrstev pomocí třídy StopWatch jsou shrnuty v grafu (obr. 32). V příloze O jsou tabulky s naměřenými daty výkonu této metody.



Ilustrace 32: Graf 13: Metoda GetRows Pq Event Trend Archive

Závěr

Hlavním úkolem této práce bylo implementovat pomocí technologií .NET aktuální aplikační vrstvu pro přístup k datům v databázi a poté porovnat výkon vrstev. Za tímto účelem byly vytvořeny vrstvy přistupující k datům pomocí technologií T-SQL, Linq to SQL a Entity Framework.

Z uvedených shrnutých výsledků implementovaných metod vyplynulo, že nejvýkonější je původní vrstva přistupující k datům pomocí technologie Xpo od firmy DevExpress. Toto rozhraní je výhodné použít ve všech metodách, které jsme v rámci této bakalářské práce implementovali. Dále vrstva využívající technologii T- SQL dávala ve většině případů lepší výsledky než vrstvy implementované technologiemi Linq to SQL a Entity Framework, které se v této práci ukázalo jako nejpomalejší. Jen v případě Main Archivu z metody GetRows se ukázala technologie Entity Framework výkonější než technologie Linq to SQL. Jazyk T-SQL byl o něco rychlejší než technologie Xpo v případě dotazů na malé množství dat.

Výkon implementovaných metod by se dal ještě zlepšit. Můžeme například vyzkoušet různou indexaci tabulek v databázi, ale technologie Xpo je na velkém množství dat natolik výkonná, že by ani toto nepomohlo dosáhnout lepších výsledků technologiemi T-SQL, Linq to SQL a Entity Framework, než jakých je dosaženo pomocí technologie Xpo od firmy DevExpress.

Literatura

[1]Mike Hotek, *Microsoft SQL Sever 2008*. Vydalo nakladatelství Computer Press, a.s.

BRNO,

2009. ISBN 978-80-251-2466-6

[2] Vidya Vrat Agarwal, James Huddleston*, Databáze v C# 2008*. Vydalo nakladatelství Computer
Press, a.s. BRNO, 2009. ISBN 978-80-251-2309-6

[3] KMB systems, *Manual SMV, SMP, SMPQ*, 2009, Liberec,
[online] http://www.kmb.cz/07/doc/SMV_SMP_SMPQ-Manual-v4-cze.pdf

[4] Paolo Pialorsi, Marco Russo, Microsoft LINQ. Vydalo nakladatelství Computer

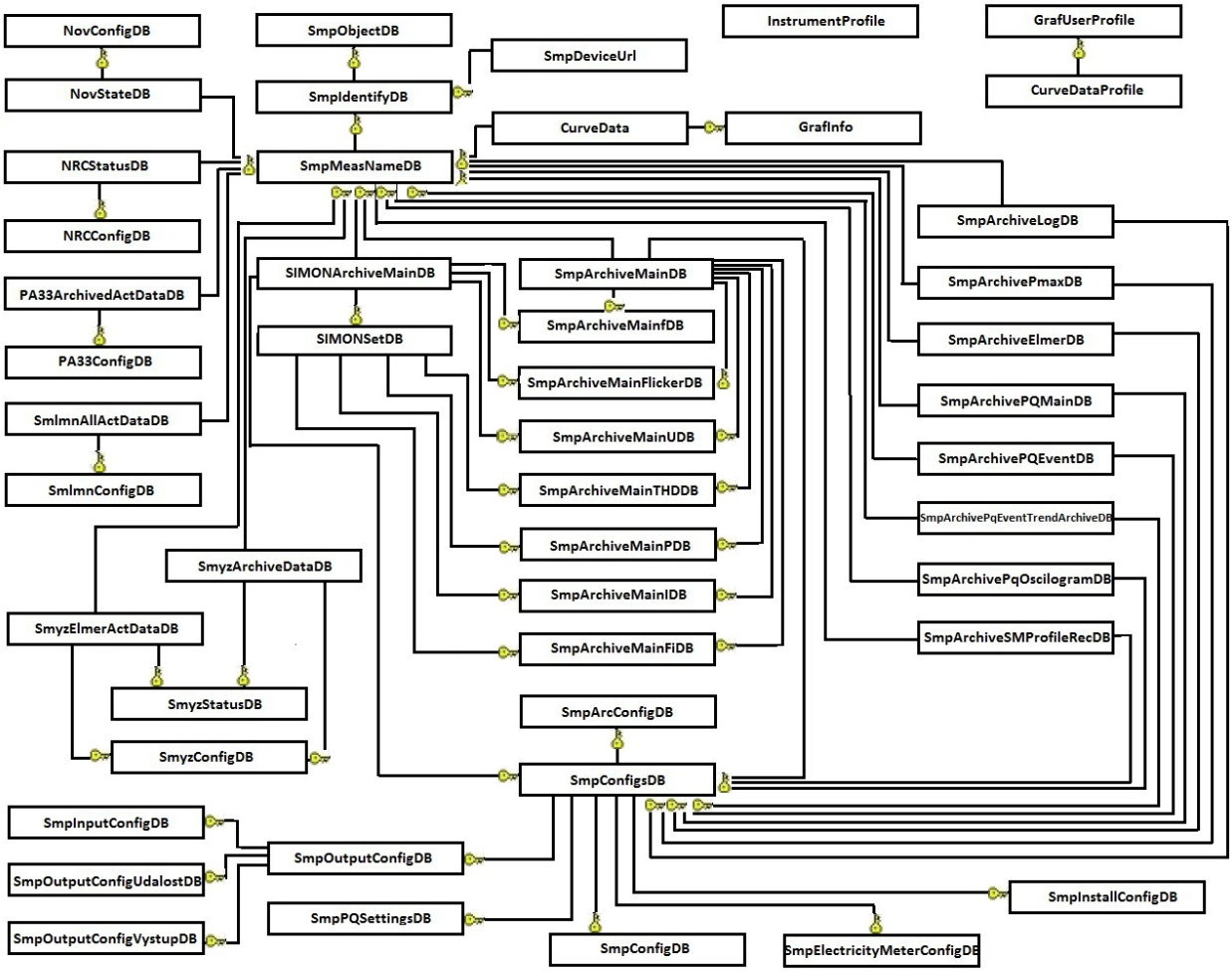
Press, a.s. BRNO, 2009. ISBN 978-80-251-2735-3

Přílohy

Příloha A:

Název Tabulky	Informace uložené v tabulce
DatabaseUpdates	Obsahuje informace o tom, kdy a na jakou verzi byl ENVIS updatován.
GrafInfo	Informace o jednotlivých grafech. Obsahující jednu a více křivek z CurveData tabulky.
GrafUserProfile	Obsahuje informace o měřených veličinách, které se mají zobrazit v grafu.
InstrumentProfile	Obsahuje XML soubory s profily na načítání hodnot z měřících přístrojů jiných společností.
NovConfigDB	Konfigurační nastavení přístroje NOVAR v době měření přísluší vždy jednomu záznamu.
NovStateDB	Obsahuje naměřená data z NOVARU.
NRCConfigDB	Konfigurační nastavení přístroje NOVAR NRC.
NRCStatusDB	Obsahuje naměřená data z NOVAR NRC.
PA33ArchivedActDataDB	Obsahuje data z přístroje PA 33
PA33ConfigDB	Konfigurační nastavení přístroje PA 33.
SIMONArchiveMainDB	Hlavní archiv přístroje SIMONPQ, obsahuje změřené hodnoty.
SIMONSetDB	Simon měří proudy po čtveřicích a může měřit až 6 čtveřic, tato tabulka obsahuje data o jednotlivých čtveřicích.
SmlmnAllActDataDB	Jedná se o archív 3 možných přístrojů SML, SMM, SMN a obsahuje změřená data z těchto přístrojů.
SmlmnConfigDB	Stažená konfigurační nastavení přístroje v době měření.
SmyzArchiveDataDB	Hlavní archív přístrojů SMY a SMZ.
SmyzConfigDB	Nastavení přístrojů SMY/SMZ, obsahuje MTN, MTP atp.
SmyzElmerActDataDB	Elektroměrové archivy přístrojů SMY/SMZ.
SmyzStatusDB	Další nastavení přístrojů SMY/SMZ.

Příloha B:



Příloha C:

Tabulka naměřených hodnot metody GetObject

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	4,09	2,2	5,73	39,55
1. čtvrteční hodnota	4,23	2,3	5,91	41,39
Střední hodnota	4,39	2,51	6,1	42,73
3. čtvrteční hodnota	4,70	2,95	6,35	44,59
Maximální hodnota	7,72	5,84	12,72	82,8

Příloha D:

Tabulka naměřených hodnot metody GetIdents

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	4,09	2,53	17,21	40,1
1. čtvrteční hodnota	4,68	2,85	17,71	41,99
Střední hodnota	5,22	3,08	18,12	43,01
3. čtvrteční hodnota	5,97	3,64	18,93	46,33
Maximální hodnota	15,22	9,74	31,5	68,87

Příloha E:

Tabulka naměřených hodnot metody GetRecords

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	3,75	2,33	8,41	40,45
1. čtvrteční hodnota	3,95	2,52	8,82	42,49
Střední hodnota	4	3,07	9,03	43,6
3. čtvrteční hodnota	4,97	3,56	9,38	45,47
Maximální hodnota	5,03	12,18	26,51	103,32

Příloha F:

Tabulka naměřených hodnot metody ExistSomeArchives

Main Archive

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	2,88	2	77,32	112,5
1. čtvrteční hodnota	2,97	2,32	78,58	116,93
Střední hodnota	3,04	2,57	79,55	119,68
3. čtvrteční hodnota	3,31	2,83	81,73	122,66
Maximální hodnota	45,79	5,35	151,36	166,44

Příloha G:

Tabulka naměřených hodnot metody ExistSomeArchives

Elmer Archiv

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	2,69	2,14	36,4	52,48
1. čtvrteční hodnota	2,93	2,22	36,86	55,47
Střední hodnota	3,09	2,34	37,09	57,52
3. čtvrteční hodnota	3,32	2,54	37,64	61,55
Maximální hodnota	12,99	10,12	54,77	103,43

Příloha H:

Tabulka naměřených hodnot metody ExistSomeArchives

Pq Oscillogram

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	2,75	2,14	14,72	39,42
1. čtvrteční hodnota	2,81	2,2	15,17	40,65
Střední hodnota	2,87	2,27	15,42	41,85
3. čtvrteční hodnota	3,15	2,58	15,92	45,35
Maximální hodnota	98,03	5,14	27,98	190,62

Příloha I:

Tabulka naměřených hodnot metody ExistSomeArchives

Pq Event Trend Archive

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	2,73	2,12	17,01	38,12
1. čtvrteční hodnota	3,03	2,23	17,31	39,7
Střední hodnota	3,13	2,47	17,47	41,34
3. čtvrteční hodnota	3,39	2,8	17,65	43,41
Maximální hodnota	12,5	6,78	33,57	61,07

Příloha J:

Tabulka naměřených hodnot metody

GetRows – Main Archive – 100 řádků

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	33,99	1521,14	30360,61	10818,93
1. čtvrteční hodnota	34,05	2824,03	31149,27	10848,79
Střední hodnota	34,56	2891,36	31469,99	10971,06
3. čtvrteční hodnota	35,48	3110,35	32089,32	11336,69
Maximální hodnota	65,79	3544,02	36351,93	11999,23

Příloha K:

Tabulka naměřených hodnot metody

GetRows – Main Archive – 1000 řádků

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	331,86	9997,21	200332,8	174177,15
1. čtvrteční hodnota	336,03	10600,56	204838,71	177359,43
Střední hodnota	342,49	11963,80	288552	181057,68
3. čtvrteční hodnota	363,35	13500,03	304487,96	181486,35
Maximální hodnota	380,58	14515,98	326585,59	199368,19

Příloha L:

Tabulka naměřených hodnot metody

GetRows – Elmer Archive – 1000 řádků

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	303	268,55	329,82	515,61
1. čtvrteční hodnota	305,48	282,47	337,95	525,99
Střední hodnota	327,10	292,33	368,77	540,27
3. čtvrteční hodnota	334,28	343,8	415,25	576,07
Maximální hodnota	340,65	405,65	450,57	591,95

Příloha M:

Tabulka naměřených hodnot metody

GetRows – Elmer Archive – 5000 řádků

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	147,91	426,3	585,16	919,76
1. čtvrteční hodnota	148,80	428,3	610,92	934,91
Střední hodnota	149,01	449,07	633,7	961,61
3. čtvrteční hodnota	153,06	519,56	689,27	975,01
Maximální hodnota	159,13	548,7	726,27	1027,51

Příloha N:

Tabulka naměřených hodnot metody

GetRows – Pq Oscillogram

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	7,18	8,51	21,73	47,52
1. čtvrteční hodnota	7,64	9,07	23,24	48,81
Střední hodnota	8,19	9,57	23,34	49,13
3. čtvrteční hodnota	9,65	10,01	24,65	52,83
Maximální hodnota	10,13	10,38	25,39	69,12

Příloha O:

Tabulka naměřených hodnot metody

GetRows – Main Archive – Pq Event Trend Archive

Čas uveden v ms	IDataSource	MDataSource	LDataSource	EFDataSource
Minimální hodnota	6,16	5,87	20,08	46,11
1. čtvrteční hodnota	6,30	6,25	21,07	48,05
Střední hodnota	6,43	7,85	21,24	48,92
3. čtvrteční hodnota	6,68	8,11	22,34	49,55
Maximální hodnota	7,98	9,76	27,32	56,01